Pushing With A Physics-Based Model

by

Huan Liu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2011

Author ....
Department of Electrical Engineering and Computer Science
August 22, 2011

Certified by ....
Tomas Lozano-Perez
Professor
Thesis Supervisor

Accepted by ....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Pushing With A Physics-Based Model

by

Huan Liu

Submitted to the Department of Electrical Engineering and Computer Science
on August 22, 2011, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Humans often push when grasping or lifting is inconvenient or infeasible, because pushing requires fewer contacts and fights against only a fraction of the object's weight. However, pushing results are hard to predict, because the physical parameters that govern the pushing motion are difficult to measure.

We derived a physics-based box pushing model and implemented a feedback-based pushing pipeline using the model. Experimental results show that our pushing model has fair predictive power and our pushing pipeline can reliably push the target to the goal. We compared our physics-based method to a minimalistic baseline pushing method and showed that our method is more accurate and reliable.

Thesis Supervisor: Tomas Lozano-Perez
Title: Professor

Acknowledgements

First, I would like to express my most sincere thanks towards Professor Tomas Lozano-Perez and Professor Leslie Kaelbling for their encouragement, patience, and guidance throughout the time I have been in the Learning and Intelligent Systems group. They have been and will always be my biggest inspiration and role models. Their modest attitudes, passionate teaching, and 7 am / weekend / Thanksgiving robot hacking sessions will always be among my favorite MIT stories.

Secondly, I would like to thank my greatest lab mates Jared Glover, Lawson Wong, Sam Davies, and Jenny Barry. Thank you for answering all my silly questions, tirelessly proof reading my thesis drafts, and most importantly, being good friends with me.

Thirdly, I would like to thank Rob Platt, Qingchun Ren, and Aaron Fryman for your technical help. Thanks to you, I got a lot more sleep.

Last but not least, I would like to thank my parents, Mandy Shen, and Mandy's mom. Thanks for taking good care of me when I didn't know when to stop and rest. Special thanks to my dad, who came all the way from the other end of the earth to cook for me and patiently wait for my thesis completion, with or without any complaint.

Contents

Chapter 1

Introduction

Pushing is a common action humans use when grasping is inconvenient or infeasible, as it needs fewer contacts and fights against only a fraction of the object's weight. On the other hand, pushing is underactuated and requires one to deal with uncertainty in the coefficients of friction and pressure distribution, which are often case dependent and hard to measure. Nevertheless, we believe that the advantage of pushing outweighs the technical difficulty. In this work, we present a physics-based pushing model and test it on a pushing pipeline in the context of a PR2 robot doing tabletop manipulation.

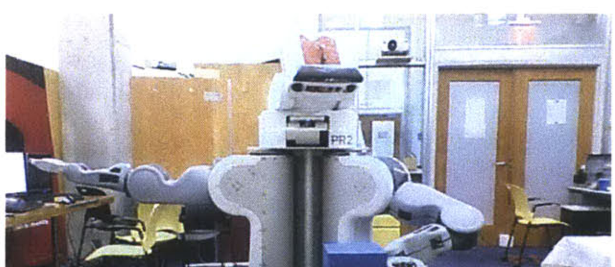

Figure 1-1: The PR2 robot pushing a box on the table.

1.1 Related Work

Existing robot manipulation planning tools such as OpenRAVE [1] and GraspIt! [2] have enabled us to perform tabletop manipulation planning using grasping as the main action primitive [3] [4]. One major problem with using grasping as the only manipulation action primitive is that grasping requires the robot to make multiple contacts with the object, which imposes rather strict kinematic constraints on the robot configuration, often requiring the robot to change its base location. Because of the more strict kinematic constraints, grasping is more prone to failure when uncertainty is involved. The success rate of finding a kinematically feasible grasp also depends on the size of the grasp set that is generated offline for the particular object.

On the other hand, the pushing action is much less constraining as it requires fewer contacts with the object. The robot can use many parts of its body to make the desired contacts with the object. In addition, for horizontal pushing, the robot often only needs to counter the friction that is a small fraction of the object's weight, allowing the robot to manipulate objects that are too heavy to be lifted. Pushing also enables manipulation with objects that are too big to be grasped.

1.1.1 Push Modeling

Mason [5] introduced mechanics for pushing and showed that using his Voting Theorem, assuming quasi-static pushing and uniform contact coefficient of friction, we could predict the rotation direction of a pushed object given the friction cone edges and line of pushing. Howe and Cutkosky [6] later presented a limit surface based model that could approximate the quasi-static motion of an object.

1.1.2 Learning-based Pushing

Christiansen, Mason and Mitchell [7] demonstrated a robot learning to move an object to different locations on a tray by tilting the tray without initial physical models. They directly mapped actions to outcomes and kept updating the mapping as new observations arrived. Over time, the mapping got closer to the underlying physical models, so prediction using the mapping became more accurate.

Walker and Salisbury [8] used a similar online learning approach to deal with the uncertainty involved in pushing. They created a manipulation map that mapped push action to object motion, which implicitly described the underlying physical parameters and avoided direct modeling. The limitation of this approach is that the learned model is not generalizable to a different object.

1.1.3 Model-based Pushing

Lynch and Mason [9] defined stable pushing as maintaining fixed contact with the object when the pusher moves. They studied the controllability of stable pushing, and proposed a planner that could find stable pushing paths among obstacles. Inside the planner, they used the limit surface model to predict pushing outcomes.

Brost [10] showed that pushing an object before grasping could reduce the object's pose uncertainty and increase grasping performance. Building on this idea, Dogar and Srinivasa [11] used push-grasps to improve grasping performance. A push-grasp pushes the object for a certain distance before grasping it. Dogar and Srinivasa [12] extended their push-grasp action to the outside region of the hand, resulting in a sweep action that could move an object out of certain region. Unlike the push-grasp that could eventually have the object roll into the hand, the uncertainty of the object's final location could be large, because the exterior of the hand could not provide such caging effect, and the limit surface approximation might be too conservative to predict the object's final location.

1.2 Problem Definition

We want to push an object from its current location *curLoc* to the goal location *goalLoc*. *curLoc* is a probability distribution of the possible current object locations. We want to get a set of push actions U, and the resulting probability distribution $goalLoc$ representing the possible final object locations. A push action u consists of an initial push contact position $pushPos$, a push direction $pushDir$, and a push distance $pushDist$. We are given a geometric model of the object, and a set of approximated physical parameters about the object, such as the coefficients of friction and the object's pressure distribution.

Box pushing In this work, we focus on the simple case of pushing a box on a flat table. We hope to illustrate the key challenges in the pushing problem and lay down ground work for future development.

1.3 Our Approach

To push an object, we use a physics-based feedback loop involving three steps: perceive, plan, and act.

Perceive The perception system processes a 3D point cloud of the environment, matches it against the known model of the object, then outputs the position and orientation of the object.

Plan The high-level planner takes in the estimated current object pose, finds a push action that will most likely move the object closest to the goal by sampling physics parameters and simulating the possible motion of the object using our physics-based pushing model.

Act The low-level motion planner translates the high-level push action into joint angle trajectories to move the robot.

We repeat the steps above until the object is close enough to the goal.

The advantage of our approach is that we model the push physics in great detail to allow accurate push simulation. As a result, one can apply parameter estimation and learning methods to tune the parameters to accurately model the real world.

Chapter 2

Physics Modeling

2.1 System Dynamics For Box Pushing via Single Contact

Quasi-static assumption We assume that the contact moves slowly so that the dynamic effects such as acceleration are negligible. In other words, we simplify the dynamics to quasi-statics, and only model the velocity of the system. As a result, the force and torque due to the contact motion balance out the friction at all times.

Friction between box and the supporting surface We make a few assumptions to model the friction between the box and its supporting surface: assume uniform coefficient of friction in the contact surface; assume uniform pressure distribution, when the box rotates, the center of rotation is at the box's center of friction; assume uniform density, the box's center of mass is its geometric center, which is directly above the center of friction. When the box rotates, friction causes a frictional rotation torque around the center of friction. When the box translates, the sliding friction opposes the motion. The torque and sliding friction are balanced by the force acting at the contact. We also approximate that the sliding friction equals the maximum static sliding friction, and the frictional rotation torque equals the maximum static frictional torque.

Contact friction cone We assume that the robot can exert any amount of force at the contact, and the only constraint is that the contact force acts within its friction cone, which is determined by the coefficient of friction between the manipulator and the box. Again, we approximate the sliding friction to be the same as the maximum static friction, and the coefficient of friction between the manipulator and the box is constant everywhere on the box. When the contact force is outside the contact friction cone, the contact slips along the box edge. If the contact force is inside the friction cone, the contact's relative position on the box does not change.

Contact force and box motion The contact force determines how the box moves. It can be decomposed into two components: the sliding-friction-opposing component and the frictional-rotation-torque-opposing component (See example in Figure 2-1). The contact force component that goes through the center of the box opposes the sliding friction between the box and its supporting surface. When this component equals the maximum static sliding friction, the box translates. The contact force component that is perpendicular to the sliding-friction-opposing component opposes the frictional torque. When this frictional-rotation-torque-opposing component generates a torque that is equal to the maximum static frictional torque, the box rotates.

The magnitude and direction of the contact force are determined by its friction cone, position on the box edge, and direction of motion relative to the box. In Section 2.1.1, we will show how to calculate the contact force in detail. Note that the contact force only acts inside its friction cone. So when the force needed to oppose a particular maximum static friction requires it to be outside of the contact's friction cone, that motion opposed by the friction cannot happen, because the contact force will only be on the edge of the friction cone and the component opposing that particular static friction will be less than the maximum static friction. Thus, different contact modes lead to different box motions.

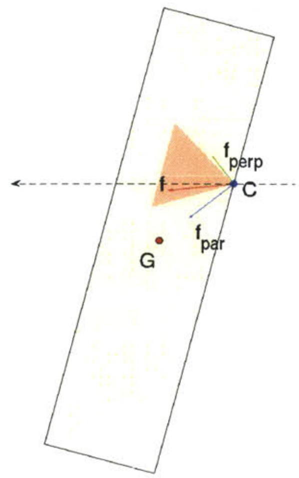

Figure 2-1: The contact force (**f** red arrow) of a contact moving horizontally from right to left (dashed black arrow) can be decomposed into two parts: the sliding-friction-opposing component ($\mathbf{f_{par}}$ blue arrow), and the frictional-rotation-torque component ($\mathbf{f_{perp}}$ green arrow). We can also see that the contact force is inside the contact friction cone (pink transparent triangle), which means that the contact force is big enough to oppose both maximum static sliding friction and maximum static frictional rotation torque, and the box will both translate and rotate.

2.1.1 Contact Force Calculation

Algorithm 2.1 calculates the contact force **f** and determines the box motion state, given the sliding friction $\mathbf{f_{SLIDING}}$, rotating friction $\mathbf{f_{ROTATING}}$, the direction of motion of contact $\mathbf{CC'}$, and the friction cone. Since we assume that the only contact force constraint is the friction cone, if the sum of the maximum static frictions lies within the friction cone, then the box will both translate and rotate. If the contact force lies outside of the friction cone, then we first make an assumption about which maximum static friction($\mathbf{f_{SLIDING}}$ or $\mathbf{f_{ROTATING}}$) the contact force **f** can oppose, then calculate the force component that does not contribute to moving the box (less than the maximum static friction), in the end, we check whether that freshly calculated force component is consistent with the assumption. If no assumption satisfies the constraint, then the box does not move.

To calculate the sliding friction $\mathbf{f_{SLIDING}}$, given the coefficient of friction between the box and its supporting surface $\mu_{SLIDING}$, mass of the box m_b, contact position C, and box's

Algorithm 2.1 Calculating contact force $\mathbf{f}$ and determining box motion state *state*, given $\mathbf{f_{SLIDING}}$, $\mathbf{f_{ROTATING}}$, $\mathbf{CC'}$, and the friction cone

```
f0 ← −f_SLIDING − f_ROTATING
if f0 is inside friction cone then
  state ← TRANSLATION_AND_ROTATION
  f ← f0
  return f
else {assuming f can only oppose f_ROTATING}
  f ← get friction cone edge based on f_ROTATING
  f_∥ ← f − (−f_ROTATING)
  if |f_∥| < |f_SLIDING| then {check if consistent with assumption}
    state ← ROTATION_ONLY
    f ← −f_ROTATING + f_∥
    return f
  else {now assume f can only oppose f_SLIDING}
    f ← get friction cone edge based on f_SLIDING
    f_⊥ ← f − (−f_SLIDING)
    if |f_⊥| < |f_ROTATING| then {check if consistent with assumption}
      state ← TRANSLATION_ONLY
      f ← −f_SLIDING + f_⊥
      return f
    else {only one possibility left}
      state ← RESTING
      return f
    end if
  end if
end if
```

center of mass position G, and the contact's direction of motion $\mathbf{CC'}$, we have:

$$\mathbf{f_{SLIDING}} = -\text{sign}(\mathbf{f'_{SLIDING}} \cdot \mathbf{CC'})\mathbf{f'_{SLIDING}},$$
$$\texttt{where } \mathbf{f'_{SLIDING}} = \mu_{SLIDING} m_b g \frac{\mathbf{GC}}{|\mathbf{GC}|}$$

To calculate the rotating friction $\mathbf{f_{ROTATING}}$, given the contact's direction of motion CC', box's center of mass position G, and maximum static frictional torque $\tau_{\mathbf{ROTATING}}$, we have:

$$\mathbf{f_{ROTATING}} = -\text{sign}((\mathbf{f'_{ROTATING}} \times \mathbf{f_{SLIDING}}) \cdot (\mathbf{GC} \times \mathbf{CC'}))\mathbf{f'_{ROTATING}}$$

$$\texttt{where } \mathbf{f'_{ROTATING}} = \frac{\tau_{\mathbf{ROTATING}}}{|\mathbf{GC}|}\left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)\frac{\mathbf{GC}}{|\mathbf{GC}|}$$

Given contact normal $\mathbf{n}$, coefficient of friction between the manipulator and the box $\mu_{SLIPPING}$, we have the condition for the contact force $\mathbf{f}$ to be inside the friction cone:

$$\cos^{-1}\left(\frac{\mathbf{f} \cdot \mathbf{n}}{|\mathbf{f}||\mathbf{n}|}\right) \leq \tan^{-1}(\mu_{SLIPPING})$$

We can get the directions of the friction cone edges $\mathbf{f_{dir}}$ ($\mathbf{f_1}$ or $\mathbf{f_2}$) by rotating the contact normal $\mathbf{n}$ by $\pm\tan^{-1}(\mu_{SLIPPING})$. To get the magnitude of the contact force on the friction cone edge, given the assumed opposable max static friction $\mathbf{f_{MAX}}$ ($\mathbf{f_{SLIDING}}$ or $\mathbf{f_{ROTATING}}$), we have:

$$|\mathbf{f}| = |\mathbf{f_{MAX}}|\left(\frac{\mathbf{f_{dir}} \cdot \mathbf{f_{MAX}}}{|\mathbf{f_{dir}}||\mathbf{f_{MAX}}|}\right)^{-1}$$

We then choose between $\mathbf{f_1}$ and $\mathbf{f_2}$ by first selecting the one with smaller magnitude, then checking to see if it is consistent with the pushing direction (because the manipulator can only exert force within ± 90 degrees of the contact's direction of motion):

$$\mathbf{f_{dir}} \cdot \mathbf{CC'} > 0$$

2.1.2 Box Kinematics

Other than resting, the box can be pushed and move in three different ways: translation-only, rotation-only, and translation-and-rotation. In this section, we will describe the kinematics for each case.

Translation-only Case

The translation-only case is straightforward. The box translates at the same velocity as the contact. See example in Figure 2-2 and Figure 2-3.

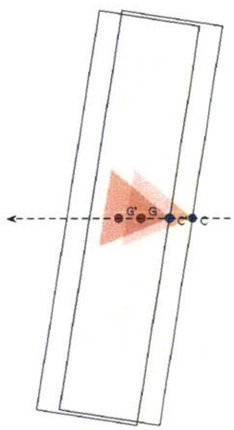

Figure 2-2: Box being pushed by a contact moving horizontally to the left. No slipping on the box edge happened as C moved to C'.

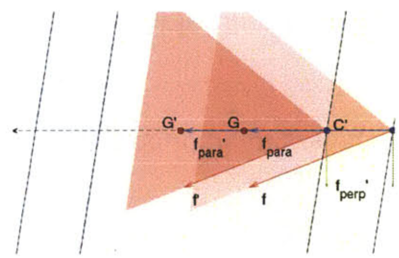

Figure 2-3: Close-up view of Figure 2-2. The contact force's direction is the same as the contact's direction of motion. The box's center of mass translates. $\mathbf{f}_{\perp}$ is too small to generate a torque that opposes the maximum static frictional torque.

Rotation-only Case

When the box only rotates, it means that the contact slips on the edge of the box. See example in Figure 2-4 and Figure 2-5. We are interested in finding the angle of rotation. Figure 2-6 is a simplified version of the diagram describing the problem.

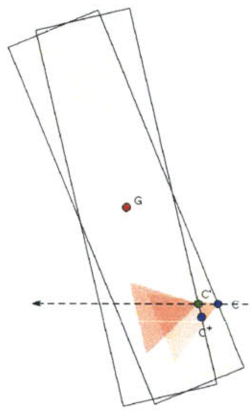

Figure 2-4: Box being pushed by a contact moving horizontally to the left. Slipping on the box edge happened as C moved to C'.

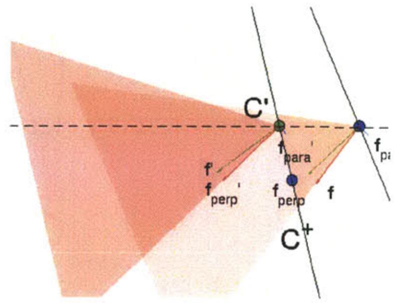

Figure 2-5: Close-up view of Figure 2-4. The contact force's direction changes as the box turns. $\mathbf{f}_{\|}$ is too small to oppose the maximum static sliding friction to translate the box.

Figure 2-6 shows that when slipping on the contact surface happens, the triangle rotates around G when it is being pushed at C. As the pusher moves from C to C', the contact moves from C to C'^{-} along the edge $\mathbf{CP}$ (if we project the new contact position back in the original triangle configuration). Given G, P, C, C', we are interested in finding the angle of rotation $\angle PGP'$, which can be calculated by:

$$\begin{aligned}\angle PGP' &= \angle PGC + \angle CGC' + \angle C'GP' \\ &= \cos^{-1}\frac{|\mathbf{PG}|}{|\mathbf{GC}|} + \cos^{-1}\frac{\mathbf{GC}\cdot\mathbf{GC'}}{|\mathbf{GC}||\mathbf{GC'}|} - \cos^{-1}\frac{|\mathbf{P'G}|}{|\mathbf{GC'}|}\end{aligned}$$

The sign of the angle of rotation is determined by applying the Voting Theorem (See Section 2.1.3).

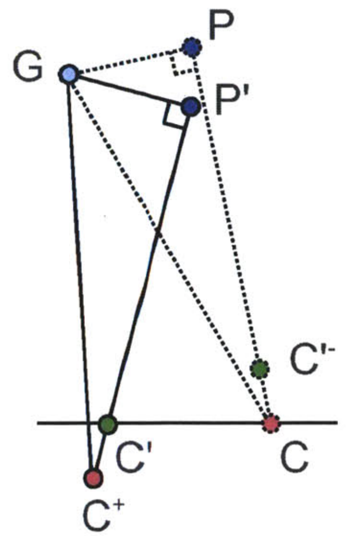

Figure 2-6: Contact C slips along the edge $\mathbf{CP}$ as it moves to C'. The object rotates around G.

Translation and Rotation Case

When the contact force is the sum of the maximum static frictions and lies within the friction cone, the box both translates and rotates. Figure 2-7 and Figure 2-8 show an example of it. We are interested in finding out the translation distance as well as the rotation angle, Figure 2-9 is a simplified version of the diagram describing the problem.

Figure 2-9 shows that when slipping on the contact surface does not happen, the triangle both translates and rotates when it is being pushed. As the pusher moves from C to C', the object translates along $\mathbf{CG}$ and rotates around the instantaneous position of G. Given G, P, C, C', we want to find G' and the rotation angle $\angle CG'C'$.

We can calculate $\mathbf{CG'}$ in the following way:

$$\begin{aligned}\mathbf{CG'} =& \frac{\mathbf{CG}}{|\mathbf{CG}|}|\mathbf{CG'}| = \frac{\mathbf{CG}}{|\mathbf{CG}|}(|\mathbf{CQ}| + |\mathbf{QG'}|)\\ =& \frac{\mathbf{CG}}{|\mathbf{CG}|}\left(|\mathbf{CC'}|\cos(\angle GCC') + \sqrt{|\mathbf{CG}|^2 - (|\mathbf{CC'}|\sin(\angle GCC'))^2}\right)\end{aligned}$$

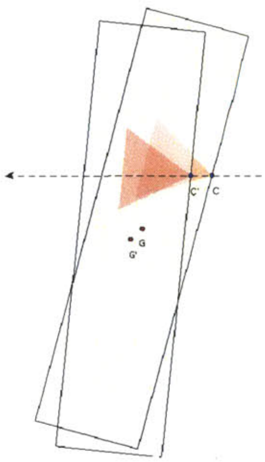

Figure 2-7: Box being pushed by a contact moving horizontally to the left. No slipping on the box edge happened as C moved to C'.

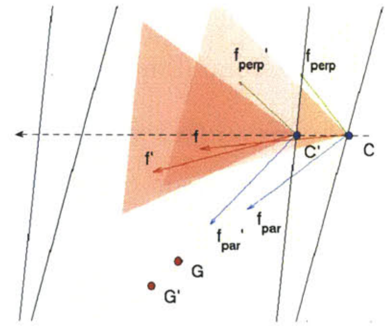

Figure 2-8: Close-up view of Figure 2-7. The contact force changes direction as the contact's direction of motion relative to the box changes due to rotation. The box's center of mass translates.

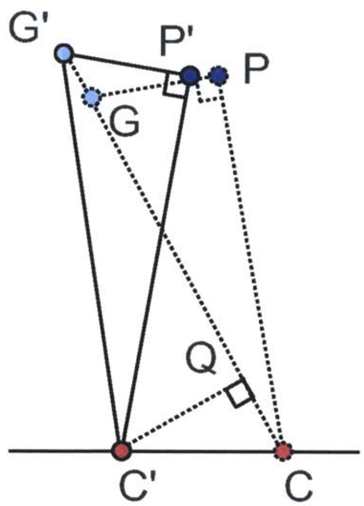

Figure 2-9: Contact C does not slip along the edge **CP** as it moves to C'. The object translates along **CG**, and at the same time, rotates around the instantaneous position of G.

Once we have G', the rotation angle $\angle CG'C' = \cos^{-1}(\frac{\mathbf{G'C}\cdot\mathbf{G'C'}}{|\mathbf{G'C}||\mathbf{G'C'}|})$. Its sign can be determined by applying the Voting Theorem (See Section 2.1.3).

2.1.3 Finding Rotation Direction

We use the "voting theorem"(Theorem 7.6 in Mechanics of Robotic Manipulation, page 152 [13]) to find the rotation direction. The voting theorem states that given pusher velocity and two edges of contact friction cone, the rotation direction can be uniquely determined by a vote of the three directed lines. Figure 2-10 is an example proven in the book.

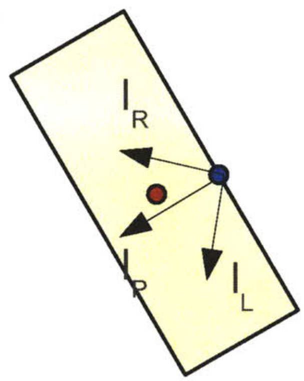

Figure 2-10: The box rotates clockwise according to the voting theorem: both line of pushing l_P and line of the left friction cone edge l_L are to the left of the box center of mass, and will rotate the box clockwise.

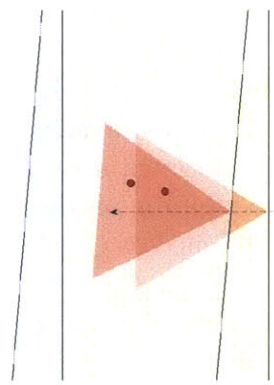

Figure 2-11: The box rotates clockwise according to the voting theorem.

If we look back at the translation-only case in Figure 2-2, the voting result is no rotation as the friction cone edges disagree and the line of pushing passes through the center of mass, resulting in a tie. In the rotation-only case in Figure 2-4, and translation-and-rotation case in Figure 2-7, all three directed lines are at the same side of the center of mass, resulting clock-wise rotation.

Chapter 3

Implementation

Given initial and goal poses, we hope to move the object from its current pose to the goal pose. Our solution is to create a feedback loop with three steps: perception, planning, and execution (Figure 3-1).

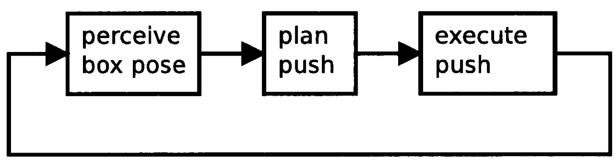

Figure 3-1: The Pushing Feedback Loop

Algorithm 3.1 is our implementation of the high-level feedback loop.

Algorithm 3.1 Feedback loop controlling pushing

$q \leftarrow$ estimateState($qInit$)
while not hasReachedGoalRegion(q,$qGoal$) **do**
    $u \leftarrow$ getControlInput(q,$qGoal$)
    execute(u)
    $q \leftarrow$ estimateState(q)
end while

We will describe the details about estimateState() in Section 3.1 Perception, getControlInput() in Section 3.2 Planning, and execute() in Section 3.3 Execution.

3.1 Perception

We use a table tracker currently being developed in our lab[14] to track the top face of the box. The tracker takes a 3D point cloud generated by a tilting laser range finder as the input. Given the height of the top face and its initial position and orientation, the tracker tries to fit a 2D polygon model of the box's top face in the point cloud. Once initialized, the tracker keeps updating the box's pose as it is pushed around. Figure 3-2 describes the perception step in the pushing feedback loop.

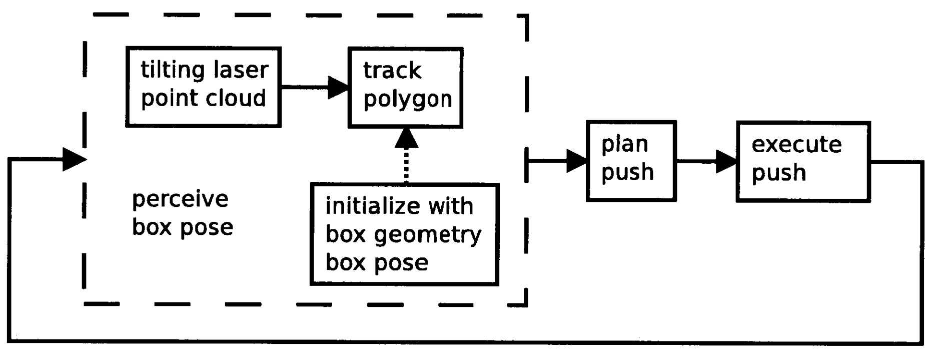

Figure 3-2: The Perception Step

The accuracy of the perception system is about 1-2 cm in translation and 1-5 degrees in rotation depending on the relative box pose to the sensor. Figure 3-3 shows the robot and the point cloud generated by the tilting laser scanner. In Figure 3-4, the blue box is the detected box top face.

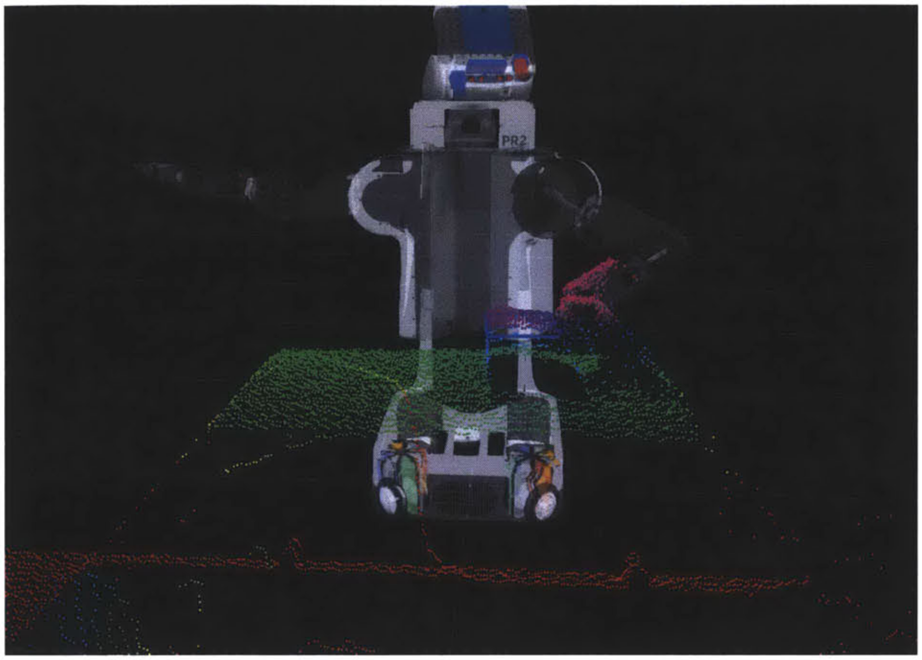

Figure 3-3: Point Cloud And Robot

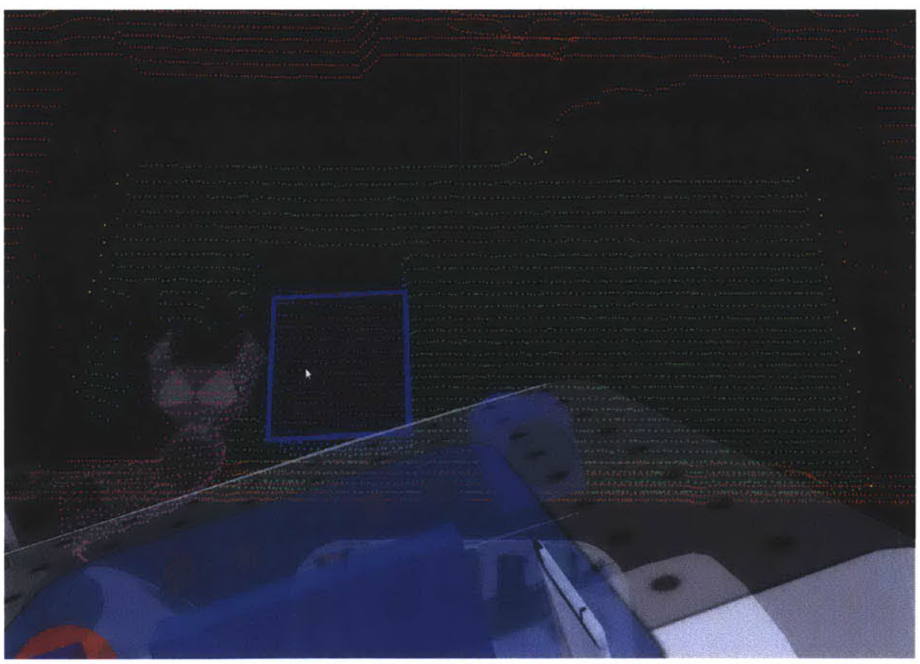

Figure 3-4: Detected Box Top Face

3.2 Planning with Perfect Perception

Our high-level push planner uses physics-based simulation to select the best action. We assume perfect perception and only consider uncertainty in the physical parameters in our planning. Figure 3-5 is a diagram describing the details of the planning step.

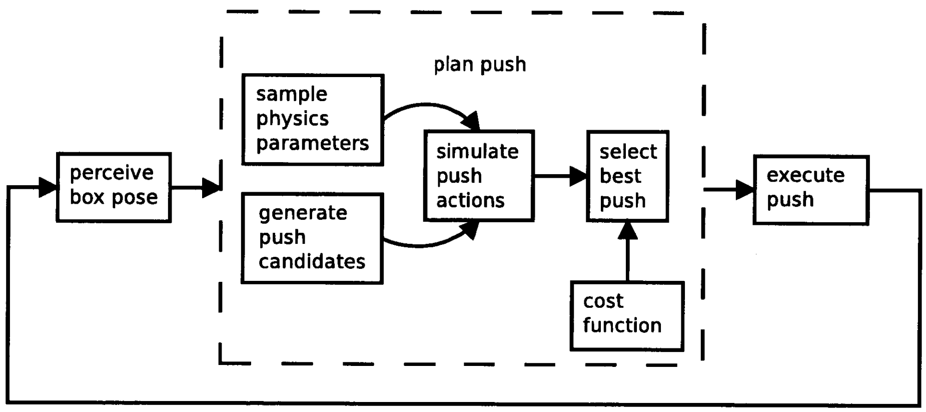

Figure 3-5: The Planning Step

Once we have the current box pose from perception, we start planning for the next push. We represent uncertainty in physical parameters using probability distributions, and we draw samples of physical parameters from those distributions. Then we apply each candidate action in simulations with all the parameter samples. We calculate the expected cost for each action over all physical parameter samples. In the end, we select the action that has the lowest expected cost.

Algorithm 3.2 is our implementation of the planning step.

getPushLocs(), getPushDirs(), and getPushDists() are called to generate the candidate push actions. generateParamSamples() generates physics parameters we use to simulate the box motion. Once we have simulation results *qOuts* for all candidate input and sampled parameters pairs, we select the best push action by calling selectBestAction(). In this section, we will explain in detail how those key functions work.

Algorithm 3.2 $u \leftarrow$ getControlInput(q,$qGoal$)

```
pushLocs ← getPushLocs(q,qGoal,NUM_PUSH_LOC)
pushDirs ← getPushDirs(NUM_PUSH_DIR)
pushDists ← getPushDists(NUM_PUSH_DIST)
controls ← constructInputs(pushLocs,pushDirs,pushDists)
params ← generateParamSamples()
qOuts ← []
for (u,p) in (controls × params) do
   qOuts.append(simulate(q,u,p))
end for

return selectBestAction(controls,qOuts)
```

3.2.1 Constructing Candidate Control Inputs

Each push action consists of push location, push direction, and push distance. In this subsection, we describe how we generate candidate push actions.

Finding candidate pushing locations Algorithm 3.3 describes the procedure we use to find candidate pushing locations.

Algorithm 3.3 $pushLocs \leftarrow$ getPushLocs(q,$qGoals$,NUM_PUSH_LOC)

```
pushRay ← getRay(q, qGoal)
intersectionLoc ← getIntersection(pushRay, boxModel2D)
pushEdge ← getEdge(intersectionLoc, boxModel2D)
return getPointsOnEdge(pushEdge, NUM_PUSH_LOC)
```

In getRay(), we construct a ray starting from q in the direction of $qGoal$ to q. Then we call getIntersection() to find the intersection between the ray and the box boundary. After that, we use getEdge() to find the edge with the shortest distance to the intersection. In the end, we return the desired number of candidate push locations NUM_PUSH_LOC by calling getPointsOnEdge(). Our implementation of getPointsOnEdge() uniformly samples 80% of the edge, excluding the 10% regions close to each vertices to avoid pushing at the corners.

Candidate pushing directions To find candidate pushing directions, we uniformly sample within ± 36 degrees from the surface normal direction. The surface normal limit of angle±36 degrees is set to prevent the rest of the robot other than the part making the desired contact from making unwanted contacts with the object. However, this limit can be removed when the robot model is introduced in the simulation or a sanity check is placed in the low-level motion planner ensuring that only the desired contact will be made during a push.

Candidate pushing distances The candidate pushing distances are represented by a set of predefined numbers in our implementation for simplicity. We can also sample a continuous distribution representing pushing distance to make the input space completely continuous.

3.2.2 Feedback-loop Modification For Pushing Straight

When we want the robot to push the box straight to its destination, the direction and location of pushing may not need to change much. Reusing previous control inputs would help reduce uncertainty in control when we do not have enough control input samples. We may reuse the previous optimal push action with a small modification illustrated in Algorithm 3.4.

Algorithm 3.4 Feedback loop reusing action

$\underline{lastBestU \leftarrow uInit}$
$q \leftarrow$ estimateState($qInit$)
while not hasReachedGoalRegion(q,$qGoal$) **do**
    $\underline{u \leftarrow \text{getControlInput}(q, qGoal, lastBestU)}$
    $\underline{lastBestU \leftarrow u}$
    execute(u)
    $q \leftarrow$ estimateState(q)
end while

Note that in addition to q and $qGoal$, getControlInput() takes in $lastBestU$, and compares it against new control input samples. $lastBestU$ provides a lower bound in control performance and ensures that the robot only picks a new push action when it is better than the old one.

3.2.3 Physical Parameters

We focus on uncertainty in the physical parameters because those parameters are hard to accurately measure in non-lab settings. In our planning, we consider the following parameters:

- $\mu_{SLIDING}$: coefficient of friction between the box and the table
- $\mu_{SLIPPING}$: coefficient of friction between the robot manipulator and the box
- $\tau_{ROTATING}$: maximum static frictional torque between the box and the table
- m: mass of the box
- cof_x: x position of the center of friction
- cof_y: y position of the center of friction

There are different ways we can model the uncertainty in the parameters. In our implementation, we choose to use uniform distributions with predetermined fixed bounds for all the parameters as we assume no prior knowledge. We could also use truncated Gaussian distributions if we know more about the object.

3.3 Execution

The goal of the execution step is to carry out the push action on the real robot. Figure 3-6 shows how the execution step works.

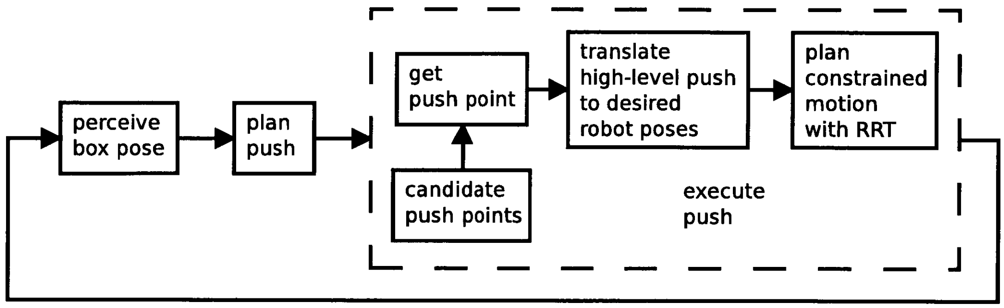

Figure 3-6: The Execution Step

To move the robot, we need to turn the high-level push action into a joint trajectory. We first find a point on the robot we can use to push the object. Then we convert the high-level push action to desired robot poses. In the end, we plan motion for the robot to move in the way that the trajectory of the chosen contact corresponds to the desired push action.

3.3.1 Finding Push Contact

For simplicity, in our implementation, we choose a fixed point (the right knuckle of our PR2's left griper shown in Figure 3-7) on our robot to be the push contact. We also define a coordinate frame with the push point as origin. We fix the orientation of the push direction in the frame, so that the robot always makes the contact with the object in the same way. Note that we can relax either the position or the orientation constraint to allow more flexible motion planning, which is an interesting topic to be explored in the future work.

3.3.2 Moving The Arm

Once we know the corresponding end effector poses of the manipulator for a push action, we need to control the robot to move its manipulator to these poses. Since we are pushing on a horizontal surface, and we do not want to exert forces in the vertical direction to change

Figure 3-7: The Push Point: the red, green and blue arrows define the local coordinates at the push point.

the pressure distribution of the object's contact surface on the table, the push motion is constrained in the x-y plane.

Using Cartesian controller One intuitive way to move the manipulator is to use a Cartesian controller, as it nicely translates motions in Cartesian coordinates into joint trajectories. However, the Cartesian controller is not aware of the potential collisions with the environment, and when the desired pose is near a singularity, the Cartesian controller cannot move the robot to the goal.

Constrained motion planning with RRT We choose to use OpenRAVE[1]'s implementation of the Rapidly-exploring Randomized Tree (RRT)[15] planning algorithm with Tangent Space Sampling introduced in [16]. In RRT's extension step, the Tangent Space Sampling method only samples configuration in the manifold specified by the user's task constraints. In addition, RRT does collision checking when it tries to connect two configurations, so the resulting trajectory will satisfy both task constraints and avoid collision with the environment. In addition, OpenRAVE's implementation of the Tangent Space Sampling avoids samples near singularity, and by allowing an error threshold, the resulting plan could avoid moving through singularity.

3.3.3 Implementation

Algorithm 3.5 is our implementation of the push execution procedure.

Coordinate transform We divide a push action into two goals: initial push goal, where the robot makes its first contact with the object; final push goal, where the robot finishes the push. In order to plan motion for the push, we first need to have the transforms for the initial and final push goals in the manipulator's end effector frame (EE), as the RRT planner takes input in the EE frame. getInitPushGoalInEEFrame(u) and getFinalPushGoalInEEFrame(u) perform coordinate transforms to the EE frame from the high level control input u.

Plan motion Once we have the goal transforms, we do two steps to get robot joint trajectories. First, we plan a motion from the robot's current pose to the initial push goal, with collision checking enabled for the target box and the rest of the environment. Then, we plan a constrained motion from the initial push goal to the final push goal, with collision checking disabled for the box, since the robot will move into the box in the box's current position.

Moving to initial push goal We call moveToHandPosition(T^{O}_{EEGoal}) to move the robot's manipulator from its current pose to the desired end effector transform in the world coordinate frame T^{O}_{EEGoal}. moveToHandPosition() is not guaranteed to return a trajectory when there exists no inverse kinematic solution for the goal pose or no path between the current and goal poses found in the maximum number of RRT iterations. So we call moveAway($T^{O}_{pInitGoal}$,P^{O}_{cInit}, P^{O}_{cFinal},$nTries$) to back up the initial push goal location in the reverse direction of pushing for up to MAX_NUM_TRIES times. If we still cannot find a

Algorithm 3.5 execute(u)

$P^O_{cInit} \leftarrow$ getInitContactLocInWorldFrame(u)
$P^O_{cFinal} \leftarrow$ getFinalContactLocInWorldFrame(u)
$T^{EE}_{p} \leftarrow$ getPushPointTransformInEEFrame()
$T^O_{pInitGoal} \leftarrow$ getPushPointInitGoalTransform(P^O_{cInit},P^O_{cFinal})
$T^O_{EEInitGoal} \leftarrow T^O_{pInitGoal} \cdot (T^{EE}_{pushPoint})^{-1}$
$T^O_{pFinalGoal} \leftarrow$ getPushPointEndGoalTransform(P^O_{cInit},P^O_{cFinal})
$T^O_{EEFinalGoal} \leftarrow T^O_{pFinalGoal} \cdot (T^{EE}_{pushPoint})^{-1}$
$T^O_{EEInitGoal} \leftarrow$ getInitPushGoalInEEFrame(u)
$T^O_{EEFinalGoal} \leftarrow$ getFinalPushGoalInEEFrame(u)
enableBoxCollisionChecking()
$nTries \leftarrow 0$
$noCollision \leftarrow$ False
while $nTries <$ MAX_NUM_TRIES **do**
    $nTries$++
    $trajdata \leftarrow$ moveToHandPosition($T^O_{EEInitGoal}$)
    if $trajdata \neq$ None **then**
        $noCollision \leftarrow$ True
    else
        $T^O_{pInitGoal} \leftarrow$ moveAway($T^O_{pInitGoal}$,P^O_{cInit}, P^O_{cFinal},$nTries$)
    end if
end while
if not $noCollision$ **then**
    disableBoxCollisionChecking()
    $trajdata \leftarrow$ moveToHandPosition($T^O_{EEInitGoal}$)
end if
executeTraj($trajdata$)
disableBoxCollisionChecking()
$trajdata \leftarrow$ constrainedMove(getCurrentEETransform(),$T^O_{EEFinalGoal}$)
executeTraj($trajdata$)

trajectory, we would disable collision checking with the box and move the hand straight to the farthest backup initial push goal. Disabling box collision checking could cause unwanted contacts with the box, but it increases the planning success rate. In future work, we could also try planning motion for a different push point if the current plan fails. For example, if we failed use the gripper knuckle to push, we could attempt to push with the gripper tip.

Constrained move Algorithm 3.6 describes the constrainedMove() implementation.

Algorithm 3.6 $trajdata \leftarrow \text{constrainedMove}(T^O_{EE0}, T^O_{EEt})$

1: $trajdata \leftarrow \text{moveToHandPositionWithConstraint}(T^O_{EEt}, constraint)$
2: **if** $trajdata = \text{None}$ **then**
3: $\quad T^O_{waypoint} \leftarrow \text{getMidPoint}(T^O_{EE0}, T^O_{EEt})$
4: $\quad trajdata \leftarrow \text{constrainedMove}(T^O_{EE0}, T^O_{waypoint}) + \text{constrainedMove}(T^O_{waypoint}, T^O_{EEt})$
5: **end if**
6: **return** $trajdata$

The constraint for table-top pushing in the manipulator frame (Figure 3-8) is that the gripper can only translate in y,z (green and blue arrows in Figure 3-8) and rotate around x (the arrow pointing down into the table). For simplicity, we assume that the distance between the initial and final push goals is small enough so that the trajectory of the push point on the robot manipulator in the constrained motion would approximate a straight line segment. We could add a sanity check to the trajectory to ensure that the push point follows the line segment closely.

Figure 3-8: The Manipulator Frame: the red (hidden), green and blue arrows define the manipulator frame.

Similar to planning for moving to the initial push goal, planning for the constrained motion to the final push goal is also likely to fail due to the planar motion constraint. Through experiments, we discovered that the RRT planner often runs out of iterations when the distance between the initial and final goals is big. By breaking the path into smaller pieces, we are able to plan constrained motion between most initial and final push goals. Since Tangent Space Sampling avoids samples in the singularity and allows an error threshold, the result trajectory might deviate from the desired straight line segment. In future work, we could implement a sanity check to see if the deviation is too big. We could also plan motion for a different push point when constrainedMove() times out.

Chapter 4

Experiments

Each of our experiments involves pushing a box multiple times to move it to a goal position about 20 cm away with the same orientation. Through experiments, we demonstrate the accuracy of our physics-based box pushing model and the robustness of our pushing pipeline. We evaluate the experiments by measuring the difference between the predicted box pose and the actual box pose after each push, the success rate of the box reaching the goal, the number of pushes it takes to reach the goal, and the average cost of each push given the cost function representing task objectives.

We want to investigate the effect of a few independent variables on the pushing performance. Namely, we consider different physical parameters such as coefficients of friction and masses, cost functions to trade-off between accuracy and safety, and sampling methods that consider uncertainty in physical parameters.

To show the performance of our pushing pipeline in an objective perspective, we compare results of our pushing pipeline against those of a baseline method that does not use a physics model.

We organize this chapter in the following way: we will first describe terminology, then we show lab setup, after that we explain experiment design, and finally we present results of our method and compare it against the baseline method.

4.1 Experiment Definition

First, we explain some experiment terminology. An *experiment* involves executing a few *pushes* using our feedback-based pushing pipeline from location A to location B on a flat table with wood surface. Figure 4-1 is a photo of our robot in the middle of an experiment. We measure performance both *push*-wise and *experiment*-wise.

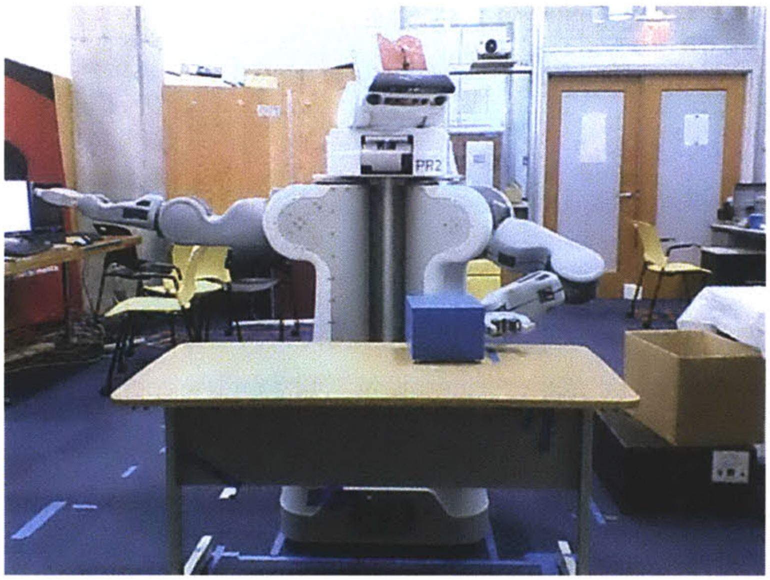

Figure 4-1: The PR2 robot pushing a box on the table.

Push A *push* consists of three steps: robot getting into the initial push pose making the initial contact with the object, robot moving the object by exerting force through the contact point, and robot arriving at the final push pose. We log the predicted object pose and measure the actual object pose after each push. Figure 4-2 demonstrates the push process.

The pictures in the left column of Figure 4-2 are visualizations of the robot's internal states of the world. In the right column are snapshots of the real world. In each of the pictures on the left, the blue box is the perceived box pose, the green box is the predicted box pose after the push (a push is visualized as the white arrow with red head), and the red box is the box pose before the push. The transparent robot in the middle-left picture is

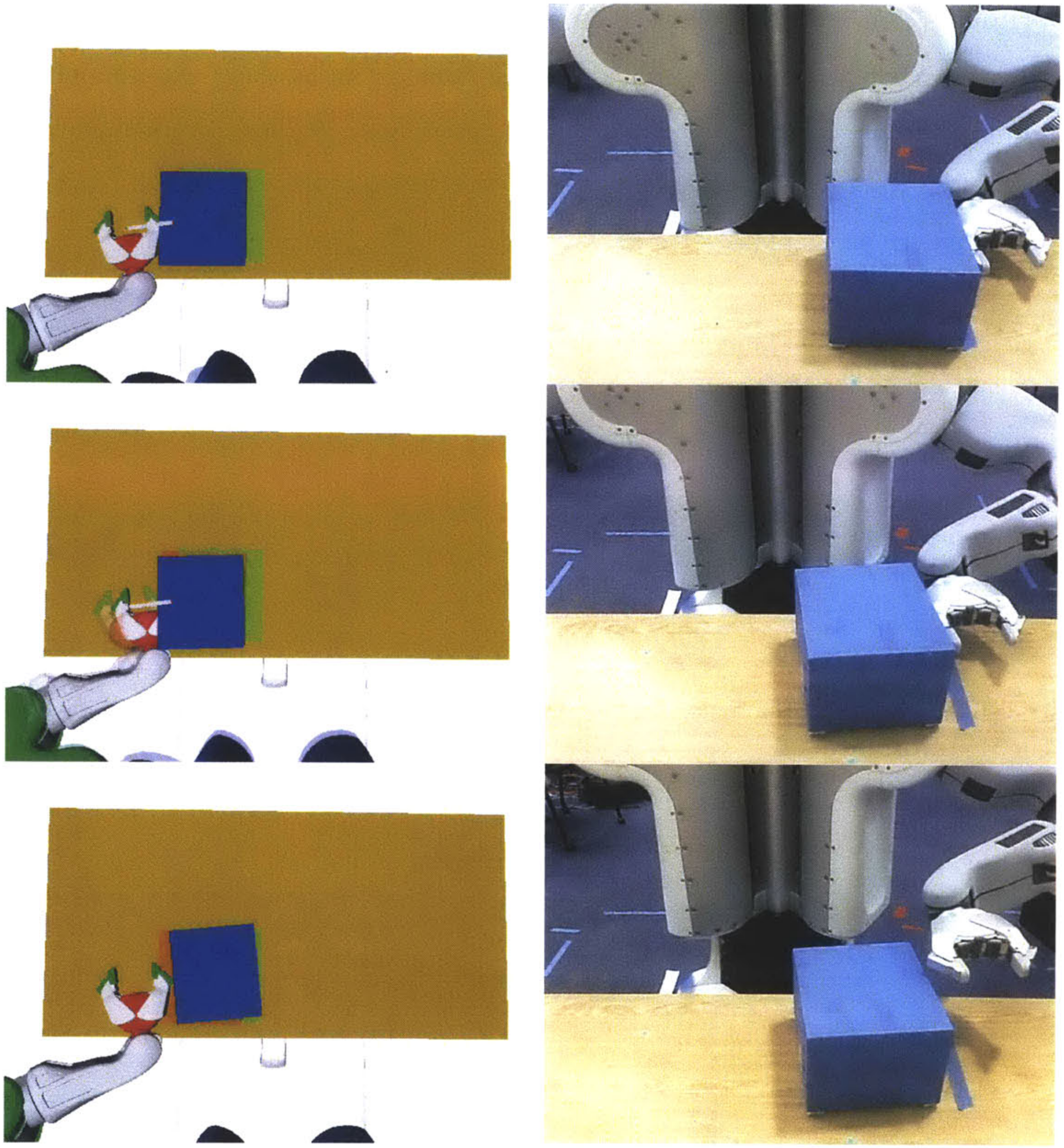

Figure 4-2: These pictures describe the first push of an experiment.

the robot's current state, the solid robot in the same picture is the future state of the robot once the motion is executed, and the right-middle photo is the result of the push action. The bottom-left picture is how the robot perceives the push result. The bottom-right photo shows the robot in its perception mode (with its torso fully raised) getting ready for the next push.

Experiment An *experiment* starts as we pass in the goal object pose into the pushing pipeline, and ends when the pushing pipeline stops. During each experiment, the robot makes a few pushes to move the object into its goal pose. We keep track of the final object pose for each experiment. Figure 4-3 shows the result of an experiment, and Figure 4-4 shows the experiment's plan.

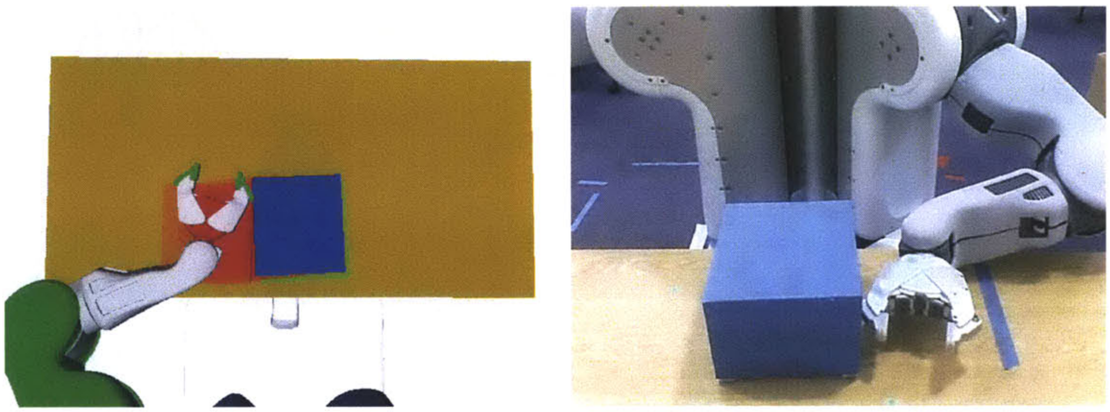

Figure 4-3: These pictures describe the end of an experiment in the robot's internal world and the real world.

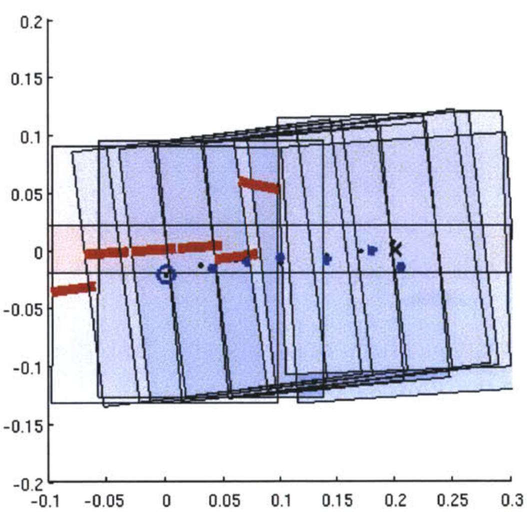

Figure 4-4: The experiment log. Both x and y axis have units in meters.

The gray boxes in Figure 4-4 are the perceived box poses before and after each push. The pushes are represented by thick red line segments. The light blue boxes are the predicted box poses. The gray boxes have black dots at their center and the blue boxes have thick blue dots indicating their center locations. The blue circle is the starting position of the box, and the black cross is the goal position.

4.1.1 Starting Condition

The box's starting pose in an experiment is no rotation, 0±1 cm in x direction, 0±2 cm in y direction (See Figure 4-4).

4.1.2 Termination Conditions

We stop the experiment under two conditions: machine termination by the pushing pipeline and manual termination.

Machine termination When the box gets close enough to the goal pose: 20 cm in x direction, 0 cm in y direction, no rotation (Figure 4-4), the pushing pipeline would declare task completion. More specifically, using the distance metric explained in Section 4.3.2, when the current pose is < 0.03 distance away from the goal pose, we stop the experiment.

Manual termination When we push the box using our pushing pipeline, if we observe that the robot has made unwanted a contact with the box (such as the case in Figure 4-5), we stop the experiment and discard the last action in our accuracy calculation. This happens when the robot attempts to push using the gripper knuckle but at an angle from the box surface normal so large that the back of the gripper or the wrist touches the box first. Since this could be prevented in planning by introducing a more strict pushing angle limit, and the error caused by such accidental contacts does not come from our model, we do not include such pushes in our error calculation. Nevertheless, we count such experiments in our overall system performance calculation.

When constrainedMove() fails to find a path in 5 minutes or returns a path that is significantly different from the desired path, we also terminate the experiment by hand. We

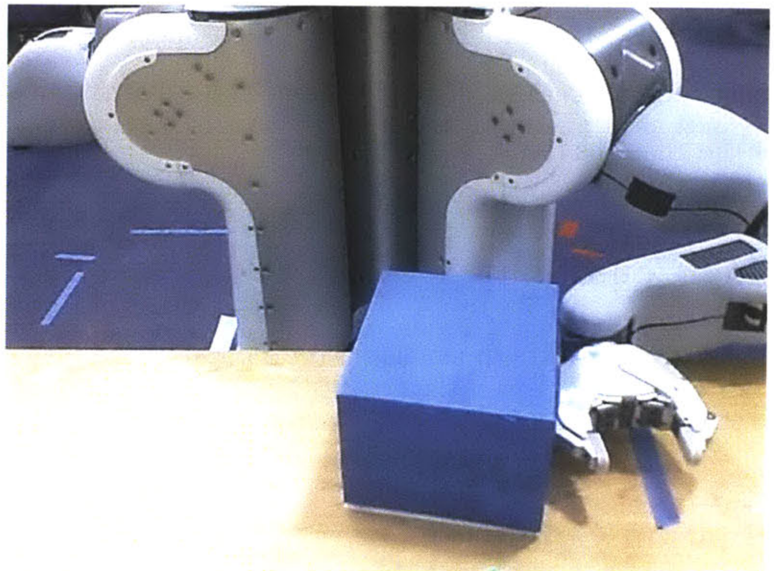

Figure 4-5: The back of gripper making unplanned contact with the box.

discard the last push action in our push accuracy calculation, but we count the final pose before the last push in our experiment success rate calculation.

We do not consider manually terminated experiments in our average experiment length calculation.

4.1.3 Candidate Push

For simplicity, we set the push distance to be 4 cm, and search over 10 uniformly sampled push positions and 10 push angles in addition to the previous best push position and angle for the next push action. Having a fixed push distance would not decrease the push resolution because we can always push at an angle closer to the object's surface angle to achieve smaller push distance. And we can get bigger push distance by concatenating pushes.

4.1.4 Baseline Method

A minimalistic baseline method is implemented to show a different pushing approach that does not use a physics model. Given the initial and goal pose, the baseline method plans a push in the following steps:

1. **Finding target edge:** Like our method, the baseline finds the target edge by extending the ray from the goal position to the initial position and the edge containing the farthest intersection is the target edge.

2. **Choosing push position:** Unlike our method, the baseline chooses the middle point of the target edge to be the push point.

3. **Choosing push direction:** Unlike our method, the push direction is determined by the direction of the line connecting the initial and goal position.

4. **Choosing push distance:** Unlike our implementation for the experiment, the push distance is the distance of the line segment connecting the initial and goal position.

Since the baseline method does not predict the motion of the box, we expect less accurate and reliable pushes generated by it than those generated by our method. In our experiments, we only implement translation and test the baseline pushing in terms of translation error.

4.2 Experiment Performance Measures

Our objective is to quantitatively measure the accuracy and robustness of our box pushing pipeline. In this section, we will explain what quantities we measure in each experiment and what we hope to learn from them.

To comprehensively measure performance, we perform experiments on boxes with different physical parameters, cost functions, and sampling methods. We concentrate on four performance criteria: accuracy for individual pushes, success rate for experiments, average number of pushes in each experiment, and average cost of pushes in each experiment.

Push Accuracy The accuracy of a push is defined as the difference between the predicted pose and the actual pose after each push. The predicted box pose is calculated by averaging the simulation results of the action for all parameter samples. We use perception result as the actual pose. To minimize perception error, we average the last five perceived box poses. In addition to the overall push accuracy, we measure and compare push accuracy given different box materials, sampling methods, and cost functions. We hope to gain insight about what variables we can focus on to improve push accuracy.

Experiment Success Rate To measure the overall performance of our pushing pipeline, we count the number of experiments where the final pose of the object is near the goal, that is less than 0.05 absolute distance away from the goal pose (0.01 distance is equivalent to 1 cm or 5 degrees, see Section 4.3.2 for the distance metric definition). We measure success rate given different box materials, sampling methods, and cost functions. We hope to learn about how each of those variables affect the high level performance.

Average Experiment Length The average experiment length measures the efficiency of our pushing pipeline. The experiment length is the number of pushes in one experiment. We record the experiment length for different experiment setups and hope to learn which variables could affect the pushing efficiency.

Average Push Cost With cost functions that represent task risks, the average push cost measures effectiveness of sampling methods. We hope to show that by sampling parameters, we increase the robustness of our pushing plan.

4.3 Experiment Variables

In this section, we will describe the experiment variables and what we can learn by varying them.

4.3.1 Physical Parameters

We vary material type and mass of the box in our experiments for both pushing methods. By doing so, we could test the performance of the pushing methods when there is uncertainty in the coefficient of frictions and mass.

Material We use four different types of contact materials of the box bottom (Figure 4-6) in our experiments: paper, acrylic, cotton and rubber. Paper, acrylic and cotton bottoms are all flat. Rubber bottom consists of four rubber felt feet at the corners. We could expect the center of friction to be near the center for all the bottoms we will use in our experiments.

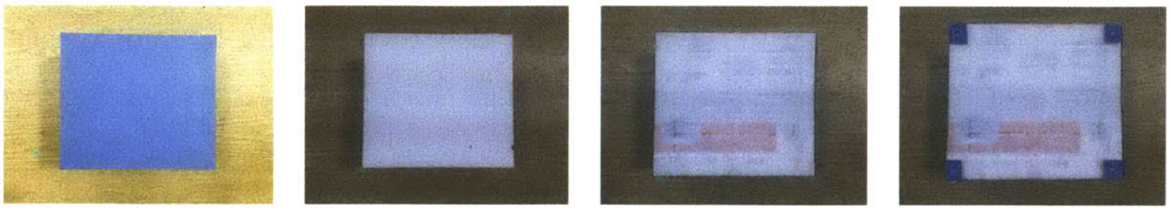

Figure 4-6: Different box bottoms

The acrylic box bottom is most slippery when being pushed while the rubber box bottom is the most sticky.

Mass Table 4.1 shows the mass of the box when different box bottoms are put on the box.

Table 4.1: Box mass of different contact material (g)

	paper	acrylic	cotton	rubber
mass	220	350	190	195

As we can see, in any case the box is relatively low mass (< 0.5kg) comparing to the 181kg robot.

4.3.2 Cost Functions

For experiments using our pushing pipeline, we use two different cost functions in planning: distance and safety. The distance cost function is used when we want the box to be pushed to the goal as fast as possible. The safety cost function is used when we want to limit the center of the box in a certain region. For example, we do not want to risk knocking the object off a shelf when we push it along the shelf's narrow deck. We hope to learn the performance of our pushing pipeline for different task objectives.

Distance The distance between q and g is calculated as follows:

$$d = \sqrt{(q_x - g_x)^2 + (q_y - g_y)^2 + ((q_\theta - g_\theta) * .01 * 180/5/\pi)^2}$$

Comparing our distance metric to the Euclidean distance metric, the difference is that we multiply our rotation error with a factor, that is, the cost of a 5 deg angle difference being the same as a 1 cm position difference to the goal. Figure 4-7 is the visualization of our distance cost function with 0 rotation error.

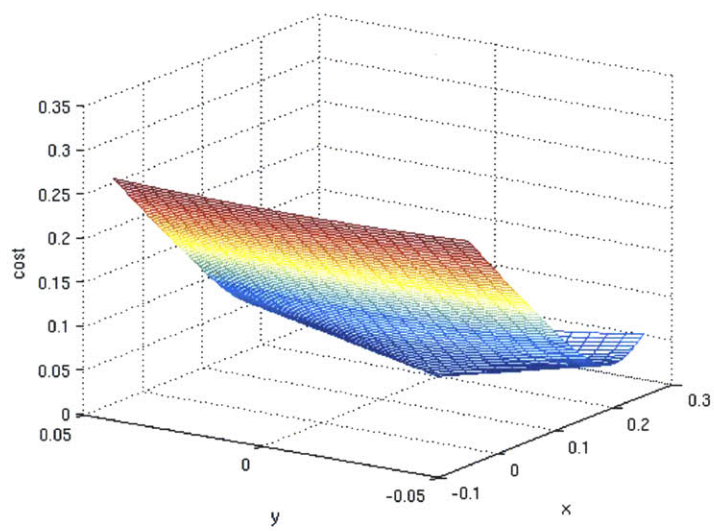

Figure 4-7: The distance cost function with 0 rotation error.

In the figure we can see that the closer the position is to the goal, the lower the cost is.

Safety The safety metric is described in the formula below:

$$d = \sqrt{(q_x - g_x)^2 + (q_y - g_y)^2 + ((q_\theta - g_\theta) * .01 * 180/5/\pi)^2} + u(|q_y - .02| > 0)$$

Note that in addition to counting the distance, the safety metric adds a penalty if the y position of the box is outside of a 4 cm band centered at the y axis. In our experiments, the center of the box starts at the origin and the goal is at 20 cm in the x direction on the y axis.

Therefore the unit cost produced by the step function $u()$ adds a significant cost compared to the distance cost. We could expect to see behavior that favors safety more than shorter distance. Figure 4-8 is a visualization of the safety cost function with 0 rotation error.

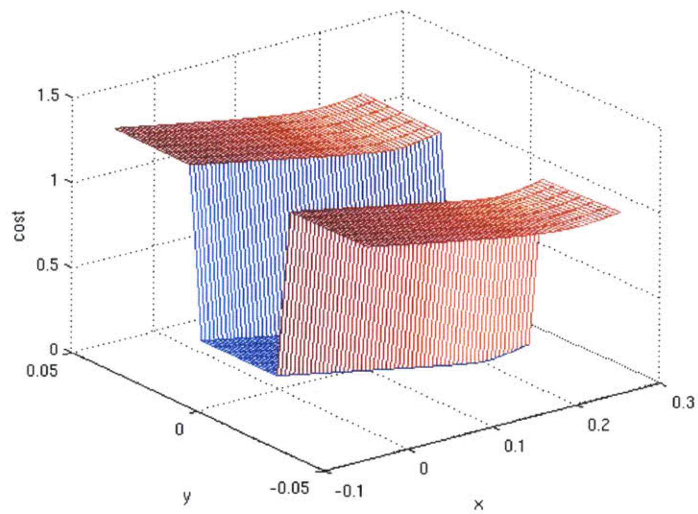

Figure 4-8: The safety cost function with 0 rotation error.

As we can see in the figure, the safety cost function has a valley in the cost surface: the cost inside the safe zone defined by $y \leq 0.02$ is a lot lower than any point outside.

4.3.3 Sampling Methods

We compare experiment results of applying our pushing method between two ways of choosing physics parameters: sampling and no-sampling. We hope to observe performance difference in terms of average cost and push accuracy. We expect that the sampling method would produce lower cost actions, because by sampling different parameters it favors actions that lead to lower expected cost. The no-sampling method might generate more accurate pushes, because in our experiment we use objects with center of friction positions close to the mean. In addition to measuring performance, we hope to justify the need of sampling when there is uncertainty in the physics parameters.

Sample from uniform distribution We use uniform distributions to represent uncertainty in the physics parameters. We assume that it is equally likely for the true parameters to be anywhere in the range we specify. Table 4.2 shows the bounds of the distributions (REC_W is the width of the box). We get the coefficients of friction range by referring to the coefficient of friction table in [13]. The mass range is defined by the masses of our box with different bottoms on. The torque and center of friction position range are set arbitrarily by hand.

Table 4.2: Bounds of uniform distributions describing the parameter uncertainty

	$\mu_{SLIDING}$	$\mu_{SLIPPING}$	$\tau_{ROTATING}$	m	cof_x	cof_y
min	.2	.4	0	.1	-REC_W/20	-REC_W/20
max	.5	.8	1	.5	REC_W/20	REC_W/20

To evaluate a push action, we first simulate the box motion using our parameter samples, then we calculate the cost for each resulting box pose, and finally we average the costs. This way, we choose the action with the best expected result. We could observe sampling choosing more conservative actions with lower cost if risk is penalized.

Note that all of our boxes have center of friction positions near mean (Section 4.3.1). So if we sample center of friction positions from our uniform distribution, we could get center of friction positions that are farther to the actual center of friction position than the mean center of friction position. As a result, the pose predictions using the center of friction samples would have a larger error. We denote experiments that sample center of friction positions *bad scenario* experiments. The performance of our pushing method in those *bad scenario* experiments would give us a lower bound measure of push accuracy.

Mean of uniform distribution We also perform experiments using the means of the uniform distributions described in Table 4.2. Because the mean method does not consider cases when true parameters are far from the mean, the optimal action chosen by the mean method could be risky. We expect to see a higher average push cost when the risk is penalized.

In terms of push accuracy, since the true center of friction positions are near the means, we denote the case of no sampling *good scenario*. The *good scenario* performance would give us a sense of the upper bound of the push accuracy.

4.4 Experiment Result

Overall, the results match our expectation. Individual pushes generated by physics based simulation have acceptable accuracy. Connecting the pushes with feedback, the experiments have a good success rate and reasonable length. In this section, we will present the result accuracy, success rate, average experiment length, and average push cost data in terms of comparisons between different sampling methods, cost functions, and contact materials. We will also show the performance comparison between physics-model based pushing pipeline and the baseline method in terms of push accuracy and success rate.

4.4.1 Accuracy

We perform experiments to measure the accuracy of individual pushes by comparing the difference between predicted box poses against actual box poses. Table 4.3 shows the average accuracy over all the pushes we have performed.

Table 4.3: Overall Push Error

trials	pushes	abs err x (cm)	abs err y (cm)	abs err θ (deg)
152	881	0.8756±0.0010	0.6811±0.0030	4.7630±0.1461
		mean err x (cm)	mean err y (cm)	mean err θ (deg)
		0.8494±0.0018	-0.2292±0.0048	-2.3549±0.4107

In Table 4.3, abs err measures *absolute* push error, while mean err measures push error. If there is no systematic error (from perception or robot actuation), we expect mean error to be random noise with mean zero.

From 881 pushes of 152 experiments, we get an average absolute position error of 1.11 cm and an average absolute rotation error of 4.76 degrees. An error of 1.11 cm in a 4 cm push is not negligible. However, it is possible that a significant portion of the error comes from perception and robot actuation, because the mean x error is very close to the absolute error instead of close to 0, which means that there might exist a constant offset in perception or actuation. Therefore, comparing experiment results by variables (such as sampling method and cost function) might give us more insight about error sources.

Sampling vs. Mean The goal of this set of experiments is to compare the push accuracy for the *bad scenario* when we have random parameters and for the *good scenario* when we have reasonable parameters. Table 4.4 shows the results of different sampling methods.

Table 4.4: Push Error of Different Sampling Methods

	trials	pushes	abs err x (cm)	abs err y (cm)	abs err θ (deg)
sampling (*bad*)	65	396	0.9410±0.0004	0.7248±0.0039	5.0363±0.1646
mean (*good*)	87	485	0.8221±0.0015	0.6454±0.0023	4.5398±0.1311

The error of the mean method is lower than that of the sampling method as we expected. The likely reason is that the box bottoms we use for experiments all have their center of friction near geometric center, which is the mean. The *bad scenario* error is about 1.19 cm and 5 degrees, which is not too much worse than the *good scenario.*

Distance vs. Safety The goal for doing experiments with different cost functions is to show how cost function could affect push accuracy. Table 4.5 contains the result.

Table 4.5: Push Error of Different Cost Functions

	trials	pushes	abs err x (cm)	abs err y (cm)	abs err θ (deg)
distance	85	491	0.8953±0.0011	0.7328±0.0029	4.9313±0.1489
safety	67	390	0.8507±0.0008	0.6160±0.0032	4.5511±0.1427

The experiments using the safety cost function have higher push accuracy than those using the distance cost function. One explanation could be that because the goal is in the middle of the safe zone, the best actions found by the safety cost function not only point away from the unsafe zone, but also point towards the goal. Since the safety cost function has a big penalty for action that cause the box to move into the unsafe zone, the actions are more conservative and more likely to move the object into the desired poses comparing to those generated by the distance cost function.

Performance for different materials Table 4.6, and Figure 4-9, 4-10 show results of experiments on the box with different bottom materials. We hope to learn about push accuracy when there is material uncertainty.

Table 4.6: Push Error of Different Materials

	trials	pushes	abs err x (cm)	abs err y (cm)	abs err θ (deg)
paper	38	221	0.8510±0.0003	0.6739±0.0019	4.2156±0.1411
acrylic	43	244	1.0085±0.0020	0.6789±0.0027	5.5678±0.2153
cotton	40	232	0.8433±0.0005	0.7461±0.0041	4.7978±0.1016
rubber	31	184	0.7694±0.0011	0.6107±0.0037	4.3094±0.1166

Among all the materials, acrylic has the highest error. The error might come from the violation of quasi-static assumption of our physics model, because we not only have acrylic with a small coefficient of friction, but also have the box with a small mass. It is easy for the robot to cause the box to accelerate and break the model prediction. On the other hand, rubber has the best accuracy, which may due to its big coefficient of friction.

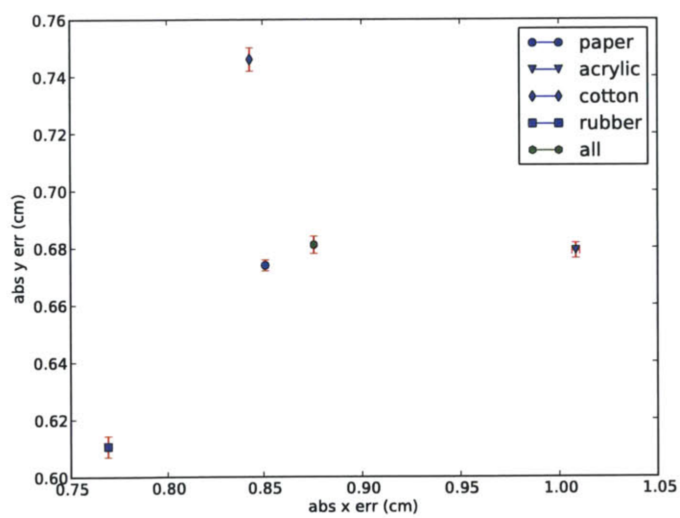

Figure 4-9: Position Error of Different Materials

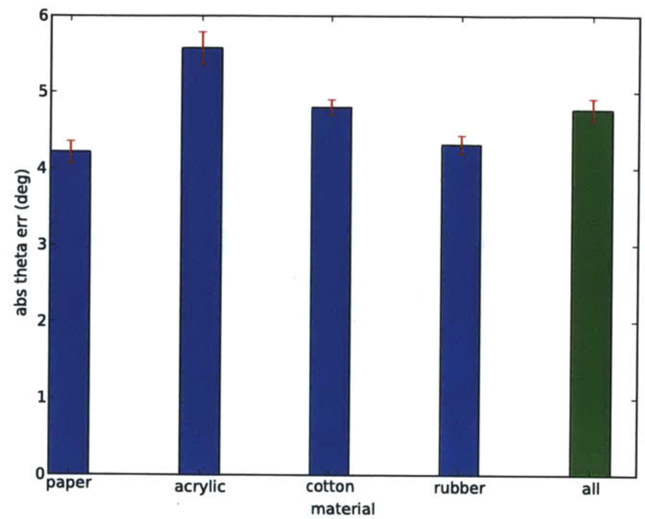

Figure 4-10: Rotation Error of Different Materials

4.4.2 Success Rate

To test the overall performance of the pushing pipeline, we carry out experiments and measure the final pose of the box and calculate its distance to goal. Table 4.7 is the result for all 152 experiments.

Table 4.7: Overall Experiment Success Rate

trials	$\leq$0.03	%	$\leq$0.05	%
152	75	49.34	112	73.68

The overall success rate is about 73% if we consider an experiment successful when the final box pose is less than 0.05 distance away from the goal. Consisting of 4 cm pushes each with 1 cm of error, the pushing pipeline has good performance thanks to the feedback mechanism.

Sampling vs. Mean We perform experiments to learn the success rate of different sampling methods. Table 4.8 shows the result data.

The sampling method has a higher success rate. However, the difference between the two is small (5% to 7%), we could say that different sampling method has small effect on the

Table 4.8: Experiment Success Rate of Different Sampling Methods

	trials	≤0.03	%	≤0.05	%
sampling	65	34	52.31	50	76.92
mean	87	41	47.13	62	71.26

success rate of a feedback based pushing pipeline.

Distance vs Safety The experiment result presented in Table 4.9 shows the effect of different cost functions on pushing pipeline success rate.

Table 4.9: Experiment Success Rate of Different Cost Functions

	trials	≤0.03	%	≤0.05	%
distance	85	33	38.82	58	68.24
safety	67	42	62.69	54	80.60

As we can see, the experiments using the safety cost function have a much higher success rate than those using the distance cost function. The explanation to this could be the same as the explanation to safety cost function producing more accurate pushes explained in the previous section.

Performance for different materials We want to test the performance of the pushing pipeline over different materials. Table 4.10 and Figure 4-11 show the result.

Table 4.10: Experiment Success Rate of Different Materials

	trials	≤0.03	%	≤0.05	%
paper	38	20	52.63	31	81.58
acrylic	44	14	31.82	28	63.64
cotton	40	22	55.00	30	75.00
rubber	31	19	61.29	23	74.19

Experiments with acrylic box bottom have the lowest success rate. We could attribute this to the slippery acrylic surface on the wood and light weight of the box breaking the quasi-static assumption of our model.

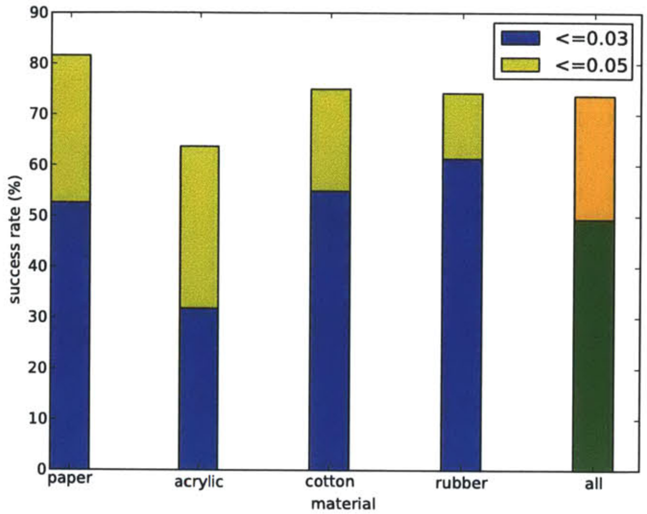

Figure 4-11: Success Rate of Different Materials

4.4.3 Average Experiment Length

We measure average experiment length to show the efficiency of our pushing pipeline. Table 4.11 shows the result for all machine terminated experiments.

Table 4.11: Overall Average Experiment Length

trials	length
130	6.1615±0.4776

For 130 experiments, it takes 6.16 pushes on average for the robot to push the box into the goal pose. The distance between the goal pose and starting pose is 0.21 and the push distance of each push action is 0.04. So ideally, the robot only needs to push 5 times. The extra push is the overhead of the pushing pipeline.

Sampling vs. Mean We want to measure the difference between the sampling methods in terms of experiment length. Table 4.12 is the result.

The mean method requires on average 0.3 less push to complete one experiment. The reason could be that the true center of friction locations being close to the mean. But the error uncertainties of $\sim$ 0.45 are large enough that this difference could be considered

Table 4.12: Average Experiment Length of Different Sampling Methods

	trials	length
sampling	59	6.3220 ± 0.4635
mean	71	6.0282 ± 0.4563

negligible.

Distance vs. Safety Table 4.13 shows the effect of different cost function on the experiment length.

Table 4.13: Average Experiment Length of Different Cost Functions

	trials	length
distance	71	6.1831 ± 0.4660
safety	59	6.1356 ± 0.4985

From the data, we can see that the safety cost function has slightly shorter average length of 0.05 steps per experiment but with a higher variance. We could say that different cost functions make little difference in the experiment length.

Performance for different materials Table 4.14 shows the average experiment length for each material.

Table 4.14: Average Experiment Length of Different Materials

	trials	length
paper	36	6.0000 ± 0.5143
acrylic	33	6.3636 ± 0.6761
cotton	34	6.0882 ± 0.3253
rubber	27	6.2222 ± 0.3333

Acrylic has most pushes in each experiment with highest variance. Since acrylic also has the lowest push accuracy (Section 4.4.1), it takes more steps to correct error.

4.4.4 Average Push Cost

We measure the average push cost in experiments with the safety cost function. Table 4.15 is the result.

Table 4.15: Overall Average Push Cost

trials	pushes	cost
67	390	0.2794

In 67 experiments, the average cost per push is 0.28, which is higher than 0.2, the distance cost between the start pose and goal pose. We can infer that in those experiments, the box has been outside of the safe zone at least once. Figure 4-12 is the log of an experiment with the initial box position being outside of the safe zone.

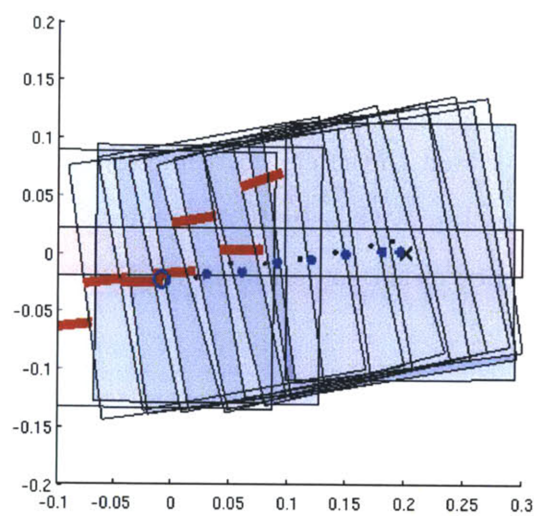

Figure 4-12: Experiment log

Sampling vs. Mean Table 4.16 shows that the sampling method generates push actions with lower average cost than the mean method.

It is true that the sampling method chooses actions with lower expected cost so that the actions are more conservative. But according to the result in Section 4.4.3, the sampling method also has longer experiments, which means that it is also possible that the lower

Table 4.16: Average Push Cost of Different Sampling Methods

	trials	pushes	cost
sampling	34	205	0.2467
mean	33	185	0.3157

average cost is due to the box spending extra time inside the safe zone with low cost. Table 4.17 shows that it is not the case.

Table 4.17: Average Time Outside Safe Zone

	trials	outside	length
sampling	34	20	1.55
mean	33	22	2.05

In Table 4.17, we look at the experiments when there is at least one observation of the box being outside of the safe zone. We calculate the average length of the box being outside. The data shows that the box spends a shorter time outside of the safe zone when it is being pushed by the actions generated by the sampling method. Therefore, we could claim that the sampling method indeed produces more conservative actions.

4.4.5 Comparing with Baseline Method

In this section, we present a performance comparison between our physics-model based pushing pipeline and the minimalistic baseline method that does not model the pushing physics. We expect to observe our method producing higher push accuracy and experiment success rate.

Accuracy Table 4.18 shows the push accuracy measurements for the baseline method. As we expected, the baseline method has higher absolute error means and variances comparing to our method (Table 4.6).

As we can see in the comparison plot (Figure 4-13) generated from the baseline result in Table 4.18 and our result in Table 4.6, our method consistently out-performs the baseline in terms of push accuracy for all materials tested in our experiments.

Table 4.18: Baseline Push Error

	trials	abs err x (cm)	abs err y (cm)
overall	40	1.2250±0.0161	2.6250±0.0055
paper	10	1.8000±0.0640	2.7000±0.0023
acrylic	10	1.1000±0.0010	3.1000±0.0054
cotton	10	1.0000±0.0000	3.0000±0.0000
rubber	10	1.0000±0.0000	1.7000±0.0023

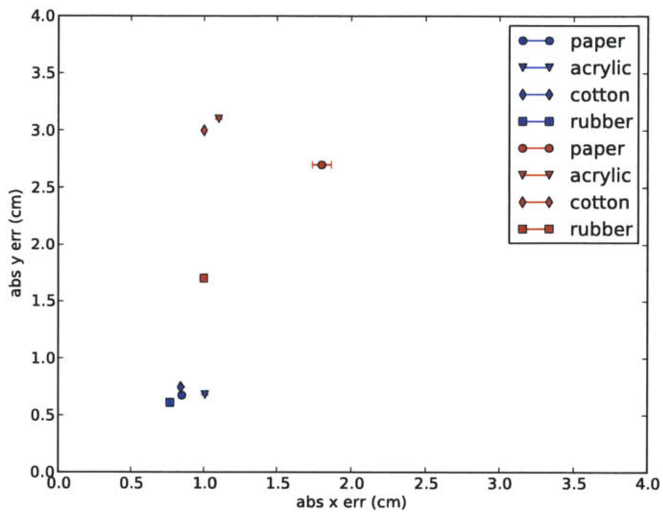

Figure 4-13: Push Position Error of Different Materials for Our Method(blue) and Baseline Method(red)

Success Rate We present the success rate result of the baseline method in Table 4.19. The baseline method's success rate for getting the object within 0.03 distance away from the goal is lower than that of our method as we expected. The fact that the success rate for getting into the 0.05 zone being much higher than ours is not surprising, either. Because we do not manually terminate the experiment in the baseline method. We believe that we could improve the success rate of our method by checking whether the manipulator will make unwanted contacts and searching for a different push when the current one fails.

Table 4.19: Baseline Experiment Success Rate

	trials	$\leq$0.03	%	$\leq$0.05	%
overall (/w physics)	152	75	49.34	112	73.68
overall (baseline)	40	14	35.00	38	95.00
paper	10	3	30.00	9	90.00
acrylic	10	1	10.00	9	90.00
cotton	10	0	0.00	10	100.00
rubber	10	10	100.00	10	100.00

Chapter 5

Discussion, Future Work and Conclusion

The experimental results demonstrate that our pushing pipeline works reliably and our physics-based model makes predictions which are more accurate than the baseline method. However, there is still a lot of room of improvement. In this section, we will describe the lessons we learned from the experiments following the order of push accuracy, experiment success rate, average experiment length, and average push cost. We will also discuss additional experiments we would like to perform to better evaluate our pushing pipeline.

5.1 Approach

5.1.1 Finding More Realistic Parameters

In the push accuracy experiments with different sampling methods, the sampling method sometimes generates actions that are less accurate than those generated by the mean method. We think that this might be because the means of our parameter distributions are close to the true parameters. So we could claim that with parameters that are close to the real parameters, our physics-based model can make good predictions. By applying parameter estimation and learning methods, we could tune our model closer to reality and plan more effectively.

5.1.2 Removing Constant Offsets

The overall push accuracy result shows that the mean error is almost the same as the absolute error in the x direction, instead of being random noise with 0 mean, which hints that there might be sources causing a constant offset in the x direction. The offset could come from perception or robot actuation. If we learn or model the offset, the push accuracy could be improved.

5.1.3 Fewer Unwanted Contacts

One major reason for an experiment using our pushing pipeline to fail to push the object to the goal is that we manually terminate the experiment when the robot makes unwanted contacts with the object. To solve this problem, we could add a sanity check in the planning step to make sure that only planned contact will be made during a push. But to improve the success rate, we might need to search for a push that satisfies our single contact constraint. In addition to trying to find a different push, we could also allow pushing with different locations and orientations on the robot, so that when push planning using one part of the robot with a particular approaching angle fails, we could try planning for pushing with a different part of the robot or same part of the robot with a different orientation.

5.1.4 Improving Push Accuracy

On average our push pipeline executes at least one more push than the ideal plan. From experiment result, we notice a strong correlation between push accuracy and experiment length. When individual pushes are more accurate, it takes fewer pushes to correct the accumulated error. So we could expect a shorter experiment when we have better push accuracy.

5.1.5 Better Cost Function

The safety cost function proves to be effectively planning pushes that keep the box inside the safe zone while moving it towards the goal. This shows that a good cost function could make our push pipeline follow the task constraints closely. Furthermore, according to the result comparing cost functions on push accuracy, cost function with hints of the solution structure could also produce better action to the problem. For example, the optimal plan from the box starting pose to the goal pose happens to be a monotonically downward trajectory towards the minimum in the cost function.

5.2 Experiments

5.2.1 Different Center of Friction Positions

We only performed experiments on boxes with center of friction positions at the center of the box bottom. We could vary the center of friction positions to test the robustness of our push pipeline when there is large center of friction position uncertainty.

5.2.2 Different Push Distance

Allowing different push distances could potentially increase the success rate of the experiments as it provides more freedom to reach the goal pose. Moreover, we could achieve higher push efficiency and have shorter experiments if we have bigger push distance.

5.2.3 Extension to Quasi Static Assumption

In our comparisons among the different push materials, the push pipeline always has the worst performance in pushing the acrylic box. The cause might be that because of its small coefficient of friction with the table and the box's small mass relative to the robot, it is very easy for the robot to accelerate the box and break the quasi-static assumption of our model. Therefore, it might be worth while to explore possible extensions to the quasi-static model.

5.2.4 Comparison Against Limit Surface Model

We would like to implement the limit-surface based pushing model and compare it against our more deliberate model. We would expect to see our model being more accurate and flexible at modeling different objects.

5.3 Conclusion

We derived a physics-based box pushing model and implemented a feedback-based pushing pipeline using our model. We performed experiments to show that our pushing model has fair predictive power and our pushing pipeline can reliably push the target object to the goal. We studied effects of experiment variables such as physical parameters, cost functions, and sampling methods, on the pushing performance. We compared our physics-based pushing method to a minimalistic baseline pushing method and showed that our method is more accurate. More work is needed to demonstrate the full power of our approach. In this thesis, we have performed a set of preliminary experiments that show the potential for improved push planning in future work.

Bibliography

[1] R. Diankov, *Automated Construction of Robotic Manipulation Programs.* PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[2] A. Miller and P. Allen, "Graspit! a versatile simulator for robotic grasping," *Robotics Automation Magazine, IEEE*, vol. 11, pp. 110 – 122, dec. 2004.

[3] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids07)*, December 2007.

[4] M. Ciocarlie, K. Hsiao, G. E. Jones, S. Chitta, R. B. Rusu, and I. A. Sucan, "Towards reliable grasping and manipulation in household environments," in *Intl. Symposium on Experimental Robotics (ISER)*, December 2010.

[5] M. Mason, "Manipulator grasping and pushing operations," in *doctoral dissertation, Electrical Engineering and Computer Science, MIT*, 1982.

[6] R. D. Howe and M. R. Cutkosky, "Practical force-motion models for sliding manipulation," *I. J. Robotic Res.*, vol. 15, no. 6, pp. 557–572, 1996.

[7] A. Christiansen, M. Mason, and T. Mitchell, "Learning reliable manipulation strategies without initial physical models," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1224 – 1230, May 1990.

[8] S. Walker and J. K. Salisbury, "Pushing using learned manipulation maps," in *ICRA*, pp. 3808–3813, 2008.

[9] K. Lynch and M. Mason, "Stable pushing: Mechanics, controllability, and planning," *International Journal of Robotics Research*, vol. 15, pp. 533–556, December 1996.

[10] R. C. Brost, "Planning robot grasping motions in the presence of uncertainty," Tech. Rep. CMU-RI-TR-85-12, Robotics Institute, Pittsburgh, PA, July 1985.

[11] M. Dogar and S. Srinivasa, "Push-grasping with dexterous hands: Mechanics and a method," in *Proceedings of 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, October 2010.

[12] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," in *Proceedings of Robotics: Science and Systems*, (Los Angeles, CA, USA), June 2011.

[13] M. T. Mason, *Mechanics of Robotic Manipulation.* MIT Press, 2001.

[14] A. Fryman and J. Glover. Personal communication, 2011.

[15] J. J. K. Jr. and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA*, pp. 995–1001, 2000.

[16] M. Stilman, "Task constrained motion planning in robot joint space," in *IROS*, pp. 3074–3081, 2007.

[17] C. Goldfeder, M. T. Ciocarlie, H. Dang, and P. K. Allen, "The columbia grasp database," in *ICRA*, pp. 1710–1716, 2009.

[18] K. Hsiao, *Relatively Robust Grasping.* PhD thesis, Massachusetts Institute of Technology, 2009.

[19] D. Berenson and S. Srinivasa, "Grasp synthesis in cluttered environments for dexterous hands," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids08)*, December 2008.

[20] D. Berenson, S. Srinivasa, and J. Kuffner, "Addressing pose uncertainty in manipulation planning using task space regions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '09)*, October 2009.

[21] K. Lynch, "The mechanics of fine manipulation by pushing," in *IEEE International Conference on Robotics and Automation*, pp. 2269–2276, 1992.

[22] K. Lynch, "Estimating the friction parameters of pushed objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 186–193, 1993.

[23] M. Mason and J. K. Salisbury, *Robot Hands and the Mechanics of Manipulation.* Cambridge, MA: MIT Press, 1985.

[24] A. Bicchi and V. Kumar, "Robotic grasping and contact: a review," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 348 –353 vol.1, 2000.

[25] J. Blythe, "An overview of planning under uncertainty," 1999.

[26] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," tech. rep., IEEE Journal of Robotics and Automation, 2004.

[27] W. Howard and V. Kumar, "On the stability of grasped objects," *Robotics and Automation, IEEE Transactions on*, vol. 12, pp. 904 –917, dec 1996.

[28] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. P. How, "Motion planning in complex environments using closed-loop prediction," in *Proc. AIAA Guidance, Navigation, and Control Conf. and Exhibit*, 2008.

[29] M. Lau, J. Mitani, and T. Igarashi, "Automatic learning of pushing strategy for delivery of irregular-shaped objects," *Evaluation*, pp. 3733–3738, 2011.

[30] D. Montana, "Contact stability for two-fingered grasps," *Robotics and Automation, IEEE Transactions on*, vol. 8, pp. 421 –430, aug 1992.

[31] V.-D. Nguyen, "Constructing force-closure grasps," in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, vol. 3, pp. 1368 – 1373, apr 1986.